Improving Compositional Generalization with Latent Structure and Data Augmentation

Linlu Qiu[1*†] Peter Shaw[1*] Panupong Pasupat[1] Paweł Krzysztof Nowak[1]
Tal Linzen[1,2] Fei Sha[1] Kristina Toutanova[1]

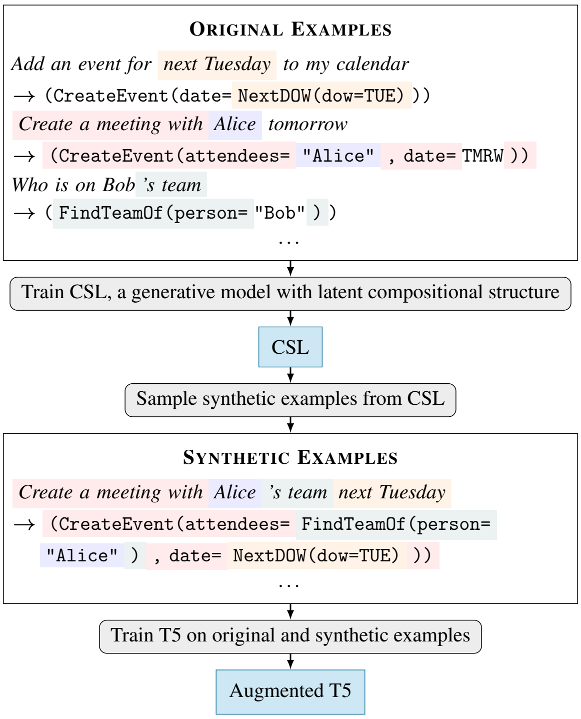
[1] Google Research [2] New York Univeristy

{linluqiu,petershaw,ppasupat,pawelnow,linzen,fsha,kristout}@google.com

Abstract

Generic unstructured neural networks have been shown to struggle on out-of-distribution compositional generalization. Compositional data augmentation via example recombination has transferred some prior knowledge about compositionality to such black-box neural models for several semantic parsing tasks, but this often required task-specific engineering or provided limited gains.

We present a more powerful data recombination method using a model called Compositional Structure Learner (CSL). CSL is a generative model with a quasi-synchronous context-free grammar backbone, which we induce from the training data. We sample recombined examples from CSL and add them to the fine-tuning data of a pre-trained sequence-to-sequence model (T5). This procedure effectively transfers most of CSL's compositional bias to T5 for diagnostic tasks, and results in a model even stronger than a T5-CSL ensemble on two real world compositional generalization tasks. This results in new state-of-the-art performance for these challenging semantic parsing tasks requiring generalization to both natural language variation and novel compositions of elements.


ORIGINAL EXAMPLES
Train CSL, a generative model with latent compositional structure
CSL
Sample synthetic examples from CSL
SYNTHETIC EXAMPLES
Train T5 on original and synthetic examples
Augmented T5

Add an event for next Tuesday to my calendar

→ (CreateEvent(date= NextDOW(dow=TUE)))

Create a meeting with Alice tomorrow

→ (CreateEvent(attendees= "Alice" , date= TMRW))

Who is on Bob 's team

→ (FindTeamOf(person= "Bob"))

...

Create a meeting with Alice 's team next Tuesday

→ (CreateEvent(attendees= FindTeamOf(person= "Alice") , date= NextDOW(dow=TUE)))

...

Figure 1: An overview of our method for compositional data augmentation with CSL, a generative model with a QCFG backbone, which is automatically induced from the training data. We show a notional set of original and synthetic examples mapping utterances to programs.

1 Introduction

Compositional generalization refers to the ability to generalize to novel combinations of previously observed *atoms*.[1] For example, we may ask a model to interpret the instruction "jump twice", when the atoms "jump" and "twice" were each observed separately during training but never in combination with each other (Lake and Baroni, 2018).

Improving compositional generalization is seen as important for approaching human-like language understanding (Lake et al., 2017; Battaglia et al., 2018) and is practically significant for real world applications, where models deployed in the wild often need to interpret new combinations of elements not well-covered by expensive and potentially skewed annotated training data (Herzig and Berant, 2019; Yin et al., 2021).

Generic neural sequence-to-sequence models have improved substantially and reached high levels of performance, particularly when combined with large-scale unsupervised pretraining and sizable in-distribution labeled data. However, these models often perform poorly on out-of-distribution compositional generalization tasks (Lake and Baroni, 2018; Furrer et al., 2020; Shaw et al., 2021).

[*] Equal contribution.

[†] Work done as part of the Google AI Residency program.

[1] Also commonly referred to as *elements* or *concepts*.

In contrast, specialized architectures with discrete latent structure (Chen et al., 2020; Liu et al., 2020; Nye et al., 2020; Herzig and Berant, 2021; Shaw et al., 2021) have made strides in compositional generalization, but without task-specific engineering or ensembling, the gains have been limited to synthetic semantic parsing tasks. Although following SCAN (Lake and Baroni, 2018) some increasingly realistic synthetic tasks such as CFQ (Keysers et al., 2020) and COGS (Kim and Linzen, 2020) have been created, and several approaches achieve good performance on these tasks, the out-of-distribution generalization ability of state-of-the-art models on real-world, non-synthetic tasks is still far from sufficient (Shaw et al., 2021; Yin et al., 2021).

Given their different strengths and weaknesses, it is compelling to combine the compositional bias of such specialized models with the greater flexibility and ability to handle natural language variation that characterizes generic pre-trained neural sequence-to-sequence models. One method for this is *data augmentation*. For example, Jia and Liang (2016) generate new training examples using example recombination via induced high-precision synchronous grammars, resulting in improvements on in-distribution and compositional splits of semantic parsing tasks. Another example is GECA (Andreas, 2020), a more general data augmentation approach that does not require task-specific assumptions. GECA achieved further gains on a larger variety of tasks, but provided limited improvements on some compositional generalization challenges.

We present a compositional data augmentation approach that generalizes these earlier methods. Training examples are recombined using the *Compositional Structure Learner* (CSL) model, a generative model with a (quasi-)synchronous context-free grammar (QCFG) backbone, automatically induced from the training data. As illustrated in Figure 1, CSL is used to sample synthetic training examples, and the union of original and synthesized examples is used to fine-tune the T5 sequence-to-sequence model (Raffel et al., 2020). CSL is more generally applicable than the method of Jia and Liang (2016), employing a generic grammar search algorithm to explore a larger, higher-coverage space of possible grammars. Unlike GECA, CSL can re-combine examples recursively and also defines a probabilistic sampling distribution over input-output pairs.

CSL builds on the NQG model of Shaw et al. (2021), a discriminative parsing model over an induced QCFG backbone, which Shaw et al. (2021) proposed to ensemble with T5. Like NQG, CSL can, on its own, address a variety of compositional generalization diagnostic tasks on synthetic datasets and achieves high precision (but limited recall) on non-synthetic compositional generalization tasks, leading to overall gains when ensembled with T5. However, CSL offers several significant improvements over NQG, allowing it to efficiently address a wider range of datasets (see §3.1). Additionally, unlike NQG which is a *discriminative* model assigning probabilities to outputs y given inputs x, CSL is a *generative* model which admits sampling from a joint probability distribution $p(x, y)$. This enables the creation of new input-output training examples.

Empirically, augmenting the training data for T5 with samples from CSL transfers most of CSL's compositional bias to T5 for diagnostic tasks (SCAN and COGS), and outperforms a T5+CSL ensemble on non-synthetic compositional generalization tasks defined by compositional splits of GeoQuery (Zelle and Mooney, 1996) and SMCalFlow (Andreas et al., 2020; Yin et al., 2021), resulting in new state-of-the-art performance on these splits.[2]

2 Background and Motivation

In this section, we discuss the problem setting common to the compositional generalization evaluations we study, and propose some general assumptions that motivate our proposed method.

Problem Setting Consider a training dataset $\mathcal{D}$ consisting of input-output pairs $\langle x, y\rangle \in \mathcal{X} \times \mathcal{Y}$, where $\mathcal{X}$ is the set of valid inputs and $\mathcal{Y}$ is the set of valid outputs. We assume that $\langle x, y\rangle \in \mathcal{D}$ are sampled from a *source* distribution $p_s(x, y)$. Our model will be evaluated on inputs from a *target* distribution $p_t(x, y)$. We make an assumption that the conditional distribution of y given x is unchanged between source and target distributions; i.e., $p_s(y|x) = p_t(y|x)$, which is also a standard assumption for domain adaptation evaluations under covariate shift. Any or all of the following may be true: $p_s(x, y) \neq p_t(x, y)$, $p_s(x) \neq p_t(x)$, $p_s(y) \neq p_t(y)$, and $p_s(x|y) \neq p_t(x|y)$.

[2] Our code is available at https://github.com/google-research/language/tree/master/language/compgen/csl.

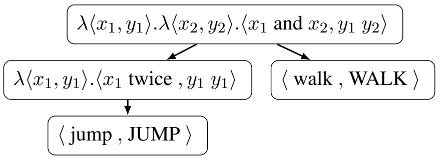

λ⟨x1, y1⟩.λ⟨x2, y2⟩.⟨x1 and x2, y1 y2⟩
λ⟨x1, y1⟩.⟨x1 twice , y1 y1⟩
⟨ walk , WALK ⟩
⟨ jump , JUMP ⟩

Figure 2: An example derivation that derives the string pair $\langle$*jump twice and walk* , *JUMP JUMP WALK*$\rangle$.

What differentiates our setting from other forms of distribution shift is the added assumption that the source and target distributions share common "atoms" (see §1). In order to translate this intuitive notion of atom sharing into formal conditions, we define a general class of models termed *derivational generative models*, based on representing atoms as *functions* which can be recombined via function application. As a modeling hypothesis, we will assume that the training and evaluation distributions can be modeled by derivational generative models that share a common set of functions, but may vary in how they assign probability to derivations formed by recombining these functions.

Derivational Generative Models A derivational generative model defines a distribution $p(x, y)$ over input-output pairs. The model contains a set of functions, $\mathcal{G}$, and a distribution over *derivations*. A derivation z can be viewed as a tree of functions from $\mathcal{G}$ which *derives* some element $[\![z]\!] = \langle x, y\rangle \in \mathcal{X} \times \mathcal{Y}$ determined by recursively applying the functions in z.[3] An example derivation is shown in Figure 2.

Given $\mathcal{X}$, $\mathcal{Y}$, and $\mathcal{G}$, we can generate a set $\mathcal{Z}^{\mathcal{G}}$ of possible derivations. We define some shorthands for important subsets of $\mathcal{Z}^{\mathcal{G}}$ for given x and y:

$$\mathcal{Z}^{\mathcal{G}}_{\langle x,y\rangle} = \{z \in Z^{\mathcal{G}} \mid [\![z]\!] = \langle x, y\rangle\}$$
$$\mathcal{Z}^{\mathcal{G}}_{\langle x,*\rangle} = \{z \in Z^{\mathcal{G}} \mid \exists y' \in \mathcal{Y}, [\![z]\!] = \langle x, y'\rangle\}$$

A derivational generative model also consists of some probability distribution $p_\theta(z)$ over the set of derivations $\mathcal{Z}^{\mathcal{G}}$, which we assume to be parameterized by θ. We define $p_{\mathcal{G},\theta}(x, y)$ in terms of $p_\theta(z)$ as:

$$p_{\mathcal{G},\theta}(x, y) = \sum\nolimits_{z\in\mathcal{Z}^{\mathcal{G}}_{\langle x,y\rangle}} p_\theta(z), \qquad (1)$$

and therefore:

$$p_{\mathcal{G},\theta}(y|x) = \frac{\sum_{z\in\mathcal{Z}^{\mathcal{G}}_{\langle x,y\rangle}} p_\theta(z)}{\sum_{z\in\mathcal{Z}^{\mathcal{G}}_{\langle x,*\rangle}} p_\theta(z)}, \qquad (2)$$

for $p_{\mathcal{G},\theta}(x) > 0$.

Discussion In general, we are interested in a set of functions that captures some knowledge of how the parts of inputs correspond to parts of outputs. If we can recover some approximation of the underlying set of functions, $\mathcal{G}$, given $\mathcal{D}$, then we could sample derivations consisting of new combinations of functions that are not observed in $\mathcal{D}$. This could potentially help us improve performance on the target distribution, since we assume that the set of functions is unchanged between the source and target distributions, and that what is varying is the distribution over derivations.

However, even assuming $\mathcal{G}$ can be exactly recovered given $\mathcal{D}$ is not sufficient to ensure that we can correctly predict the most likely y given x according to the true $p(y|x)$ (shared between source and target distributions) for $x \sim p_t(x)$.[4] We must also assume that there exists a parameterization of $p_\theta(z)$ such that when we estimate $\hat{\theta}$ given $\mathcal{D}$, $p_{\mathcal{G},\hat{\theta}}(y|x)$ sufficiently approximates the true $p(y|x)$ for $x \sim p_t(x)$. We hypothesize that conditional independence assumptions with respect to how $p_\theta(z)$ decomposes across the function applications in z can be helpful for this purpose. In particular, such assumptions can enable "reusing" conditional probability factors across the exponential space of derivations, potentially improving transfer to the target distribution.

With this intuition in mind, in §3 we propose a specific class of functions for $\mathcal{G}$ based on (quasi-)synchronous context-free grammars, as well as a factorization of $p_\theta(z)$ with strong conditional independence assumptions.

3 Proposed Method

As shown in Figure 1, our method consists of two stages. First, we induce our generative model, CSL, from training data (§3.1). Second, we sample synthetic examples from the generative model and use them to augment the training data for a sequence-to-sequence model (§3.2).

[3] Formally, let $\mathcal{G}$ be a set of partial functions over some set of elements. If $f \in \mathcal{G}$ is a constant function, then f is a ground term. If $f \in \mathcal{G}$ has arity k, and $a_1, \cdots, a_k$ are ground terms, then $f(a_1, \cdots, a_k)$ is a ground term. We define a *derivation* as a ground term that generates an element $\in \mathcal{X} \times \mathcal{Y}$.

[4] One special case is where $|\mathcal{Z}^{\mathcal{G}}_{\langle x,*\rangle}| = 1$ for all x. In this case, every x has exactly one unique derivation and $p_{\mathcal{G}}(y|x)$ is deterministic given $\mathcal{G}$ and does not depend on θ, and therefore recovering $\mathcal{G}$ *is* sufficient.

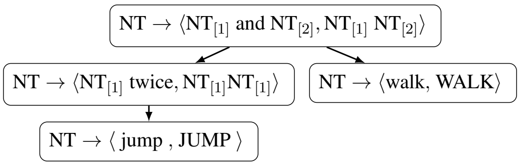

NT → ⟨NT[1] and NT[2], NT[1] NT[2]⟩
NT → ⟨NT[1] twice, NT[1]NT[1]⟩
NT → ⟨walk, WALK⟩
NT → ⟨ jump , JUMP ⟩

Figure 3: The example derivation of Figure 2 using QCFG notation.

3.1 Compositional Structure Learner (CSL)

CSL can be viewed as a derivational generative model, as defined in §2, where the set $\mathcal{G}$ of recursive functions is defined by a (quasi-)synchronous context free grammar (QCFG).[5]

We first describe the grammar formalism and the parameterization of our probabilistic model. Then we describe our two-stage learning procedure for inducing a grammar and learning the model parameters. CSL builds on the NQG model of Shaw et al. (2021), with several key differences discussed in the following sections:

- Unlike NQG, which is discriminative, CSL is a *generative* model that admits efficient sampling from the joint distribution $p(x, y)$.
- CSL enables a more expressive set of grammar rules than NQG.
- CSL offers a more computationally efficient and parallelizable grammar induction algorithm and a more efficient parametric model, allowing the method to scale up to larger datasets such as SMCalFlow-CS.

See section 4.4 for experiments and analysis comparing the components of CSL and NQG.

3.1.1 Grammar Formalism

An example QCFG derivation is shown in Figure 3, and the notation for QCFGs is reviewed in Appendix B.1. Notably, the correspondence between rules over input and output strings in QCFGs is akin to a homomorphism between syntactic and semantic structures, commonly posited by formal theories of compositional semantics (Montague, 1970; Janssen and Partee, 1997). We restrict our grammars to have only a single unique nonterminal symbol, NT. In constrast to standard synchronous context-free grammars (SCFGs), our grammars can be *quasi-synchronous* (Smith and Eisner, 2006) because we allow a one-to-many alignment between non-terminals.[6] Unlike the formalism of Shaw et al. (2021), which limited rules to contain $\leq$ 2 non-terminals, in the current work the maximal number of non-terminals is a configurable parameter,[7] which enables inducing grammars with higher coverage for certain datasets.

3.1.2 Probabilistic Model

We factorize the probability of a derivation in terms of conditional probabilities of sequentially expanding a rule from its parent. Formally, let r denote a rule expanded from its parent rule r_p's $NT_{[i]}$ non-terminal (or a special symbol at the root of the derivation tree).[8] We assume conditional independence and factorize the probability of z as:

$$p_\theta(z) = \prod_{r, r_p, i \in z} p_\theta(r | r_p, i) \tag{3}$$

This non-terminal annotation with context from the tree is akin to parent annotation or other structure conditioning for probabilistic context-free grammars (Johnson, 1998; Klein and Manning, 2003).

Using independent parameters for each combination of a rule, its parent, and non-terminal index may lead to overfitting to the training set, limiting our ability to generalize to new combinations of rule applications that are needed for compositional generalization. We therefore factor this distribution using a soft clustering into a set of *latent states* $\mathcal{S}$ representing parent rule application contexts:

$$p_\theta(r | r_p, i) = \sum_{s \in \mathcal{S}} p_\theta(r | s) p_\theta(s | r_p, i) \tag{4}$$

where $p_\theta(s | r_p, i) \propto e^{\theta_{r_p, i, s}}$ and $p_\theta(r | s) \propto e^{\theta_{s,r}}$ and the θs are scalar parameters.[9] The number of context states $|\mathcal{S}|$ is a hyperparameter. While these conditional independence assumptions may still be too strong for some cases (see Appendix C.2), we find them to be a useful approximation in practice.

[5] QCFG rules can be interpreted as functions which are limited to string concatenation. For notational convenience, we will therefore treat $\mathcal{G}$ as a set of QCFG rules in §3.

[6] Concretely, for a rule $NT \rightarrow \langle \alpha, \beta \rangle$, a non-terminal in α can share an index with more than one non-terminal in β. This is important for datasets such as SCAN, as it allows rules such as NT → ⟨NT$_{[1]}$twice, NT$_{[1]}$NT$_{[1]}$⟩ which enable repetition.

[7] We find that 4 is a computationally tractable choice for the datasets we study.

[8] Using the example derivation from Figure 3, for the rule application $r =$ NT → ⟨walk, WALK⟩, we have $r_p =$ NT → ⟨NT$_{[1]}$ and NT$_{[2]}$, NT$_{[1]}$NT$_{[2]}$⟩ and the expansion probability for that rule application is $p(r | r_p, 2)$.

[9] The full definition of these terms is in Appendix B.2.

We also optionally consider a task-specific output CFG, which defines valid output constructions.[10]

3.1.3 Learning Procedure

A principled method to estimate $\mathcal{G}$ and θ given $\mathcal{D}$ would be to find the MAP estimate based on some prior, $p(\mathcal{G}, \theta)$, that encourages compositionality:

$$\arg\max_{\mathcal{G},\theta} \; p(\mathcal{G}, \theta) \times \prod_{\langle x,y \rangle \in \mathcal{D}} p_{\mathcal{G},\theta}(x, y) \qquad (5)$$

However, since optimizing $\mathcal{G}$ and θ jointly is computationally challenging, we adopt a two-stage process similar to that of Shaw et al. (2021). First, we learn an unweighted grammar using a surrogate objective for the likelihood of the data and a compression-based compositional prior that encourages smaller grammars that reuse rules in multiple contexts, inspired by the Minimum Description Length (MDL) principle (Rissanen, 1978; Grunwald, 2004). We describe the induction objective and algorithm in §3.1.4. Second, given an unweighted grammar $\mathcal{G}$, we optimize the parameters θ by maximizing the log-likelihood of $p_{\mathcal{G},\theta}(x, y)$, as defined by Eq. 1, using the Adam optimizer (Kingma and Ba, 2015).[11]

3.1.4 Grammar Induction Algorithm

Our method for inducing a QCFG is based on that of Shaw et al. (2021), but with several modifications, which improve the computational scalability of the algorithm as well as the precision and coverage of the induced grammar. We analyze the relative performance of the two algorithms in §4.4.

Objective The main idea of the grammar induction objective, $L_{\mathcal{D}}(\mathcal{G})$, is to balance the size of the grammar with its ability to fit the training data:

$$L_{\mathcal{D}}(\mathcal{G}) = \sum_{NT \rightarrow \langle \alpha,\beta \rangle \in \mathcal{G}} |\alpha| + |\beta| - c_{\mathcal{D}}(\alpha, \beta), \quad (6)$$

where $|\cdot|$ is a weighted count of terminal and nonterminal tokens (the relative cost of a terminal vs. nonterminal token is a hyperparameter) and

$$c_{\mathcal{D}}(\alpha, \beta) = k_\alpha \ln \hat{p}_{\mathcal{D}}(\alpha|\beta) + k_\beta \ln \hat{p}_{\mathcal{D}}(\beta|\alpha) \quad (7)$$

where k_α and k_β are hyperparameters and $\hat{p}_{\mathcal{D}}(\alpha|\beta)$ is equal to the fraction of examples $\langle x, y \rangle \in \mathcal{D}_{train}$ where α "occurs in" x out of the examples where β "occurs in" y, and vice versa for $\hat{p}_{\mathcal{D}}(\beta|\alpha)$.[12] The correlation between α and β as measured by the $\hat{p}$ terms provides a measure related to how well the rule fits the training data. We use sampling to optimize the computation of $\hat{p}_{\mathcal{D}}$ for larger datasets. While conceptually similar to the objective used by NQG, we found that CSL's objective is more efficient to compute and can be more effective at penalizing rules that lead to lower precision.[13]

Initialization To initialize $\mathcal{G}$ we add a rule $NT \rightarrow \langle x, y \rangle$ for every $\langle x, y \rangle \in \mathcal{D}$. We also optionally add a set of seed rules, such as $NT \rightarrow \langle x, x \rangle$ where a terminal or span x is shared between the input and output vocabularies. For details on seed rules used for each dataset, see Appendix A.

Greedy Algorithm Following the initialization of the set of rules $\mathcal{G}$, we use an approximate parallel greedy search algorithm to optimize $L_{\mathcal{D}}(\mathcal{G})$, while maintaining the invariant that all examples in $\mathcal{D}$ can be derived by $\mathcal{G}$.

At each iteration, the algorithm considers each rule r in the current grammar in parallel. The algorithm determines a (potentially empty) set of candidate actions for each r. Each candidate action consists of adding a new rule to the grammar that can be combined with an existing rule to derive r, enabling r to be removed. Certain candidate actions may enable removing other rules, too. The algorithm then selects the candidate action that leads to the greatest improvement in the induction objective, if any action exists that leads to an improvement. The selected actions are then aggregated and executed, resulting in a new set of rules. The algorithm continues until no further actions are selected, or a maximum number of steps is reached. The detailed implementation of the greedy algorithm is detailed in Appendix B.

3.1.5 Inference Procedure

While the primary goal of CSL is to be used to sample new examples for data augmentation (discussed

[10] The outputs for several of the tasks we study consist of executable programs or logical terms, for which we can assume the availability of a CFG for parsing. Details are in Appendix A.

[11] To optimize θ efficiently, we use a variant of the CKY algorithm (Cocke, 1969; Kasami, 1965; Younger, 1967) to determine the set of derivations, represented as a parse forest, and use dynamic programming to efficiently sum over this set.

[12] By α "occurs in" x, we mean that there exists some substitution for any non-terminals in α such that it is a substring or equal to x.

[13] For example, CSL's objective enables our algorithm to induce a "clean" 20 rule grammar for SCAN, while using our algorithm with the objective of NQG leads to grammars with additional spurious rules for SCAN.

next), we can also use CSL as a discriminative parsing model, by using a variant of the CKY algorithm to find the highest scoring derivation z that maximizes Eq. 3 for a given input x. We then output the corresponding y if it can be derived by the given output CFG, or if no output CFG is provided.

3.2 Data Augmentation

We synthesize a configurable number of examples by sampling from the learned generative model, CSL.[14] To generate a synthetic example (x, y), we use forward sampling: we start from the single NT symbol and sample recursively to expand each nonterminal symbol with a rule, based on $p_\theta(r|r_p, i)$ defined by Eq. 4.[15]

Given that generic sequence-to-sequence models perform poorly on length extrapolation (Newman et al., 2020), we optionally bias our sampling to favor deeper derivations. We achieve this by adding a bias $\delta > 0$ to $\theta_{t,r}$ for any rule r that contains more nonterminals than our configurable threshold.

We fine-tune T5 on the union of the original training data and the synthesized data. Following Jia and Liang (2016), we ensure an approximately equal number of original and synthesized examples are used for training. We achieve this by replicating original or synthetic examples as needed.

4 Experiments and Analysis

In this section, we comparatively evaluate and analyze our main proposed method, T5+CSL-Aug., which uses CSL to generate examples for augmenting the training data of T5.

4.1 Datasets

We evaluate our approach on both synthetic benchmarks designed for controlled assessments of compositional generalization, and non-synthetic evaluations, which introduce the additional challenge of handling natural language variation. For example, some words in the test data might never appear during training. Further details on datasets and preprocessing are in Appendix A.

SCAN The SCAN dataset contains navigation commands paired with action sequences. We consider three compositional data splits from Lake and Baroni (2018): the *jump* and *turn left* splits (where a new primitive is used in novel combinations), and the *length* split. We also consider the *MCD* splits from Keysers et al. (2020) created by making the distributions of compositional structures in training and test data as divergent as possible.

COGS The COGS dataset (Kim and Linzen, 2020) contains sentences paired with logical forms. We use the generalization test set, which tests generalization to novel linguistic structures. As SCFGs cannot handle logical variables (Wong and Mooney, 2007), we convert the outputs into equivalent variable-free forms.

GeoQuery GeoQuery (Zelle and Mooney, 1996; Tang and Mooney, 2001) contains human-authored questions paired with meaning representations. We report results on the standard data split as well as three compositional splits based on those introduced in Shaw et al. (2021): the *template* split (where abstract output templates in training and test data are disjoint (Finegan-Dollak et al., 2018)), the *TMCD* split (an extension of MCD for non-synthetic data), and the *length* split.[16]

SMCalFlow-CS Yin et al. (2021) proposed a *compositional skills* split of SMCalFlow (Andreas et al., 2020) that contains single-turn sentences from one of two domains related to creating calendar events or querying an org chart, paired with LISP programs. The single-domain (S) test set has examples from a single domain, while the cross-domain (C) test set has sentences that require knowledge from both domains (e.g., "create an event with my manager"). Only a small number of cross-domain examples (8, 16, or 32) are seen during training.

4.2 Baselines

Our primary goal is to evaluate T5+CSL-Aug. in comparison to T5 and T5+GECA, a method augmenting training data with GECA which is prior state of the art for data augmentation (Andreas, 2020).[17] Details and hyperparameters for the

[14] For all experiments, we sample 100,000 synthetic examples unless otherwise indicated.

[15] We ensure the sampled y can be generated by a CFG defining valid outputs, if one is provided for the given task, by sampling from the intersection of $\mathcal{G}$ and the provided output CFG.

[16] For GeoQuery, to reduce variance due to small dataset sizes, we average all results over 3 runs. For the Template and TMCD splits we additionally average over 3 splits generated with different random seeds. Variance is reported in Appendix C.8

[17] For all experiments with T5, we show results for T5-Base (220M parameters). Prior work found T5-Base to perform best on the compositional splits of SCAN and GeoQuery (Furrer et al., 2020; Shaw et al., 2021). We found a similar trend for COGS and SMCalFlow-CS after evaluating T5-Large and

	SCAN				COGS	GeoQuery				SMCalFlow-CS					
System	Jump	Left	Len.	MCD	Gen.	Std.	Templ.	TMCD	Len.	8-S	8-C	16-S	16-C	32-S	32-C
NQG-T5	**100.0**	**100.0**	**100.0**	**100.0**	97.9	**92.9**	84.2	71.4	53.9	—	—	—	—	—	—
SpanBasedSP	—	—	—	—	—	78.9	76.3	56.5	53.9	—	—	—	—	—	—
LeAR	—	—	—	—	97.7	—	—	—	—	—	—	—	—	—	—
C2F	—	—	—	—	—	—	—	—	—	—	—	83.0	40.6	83.6	54.6
C2F+SS	—	—	—	—	—	—	—	—	—	—	—	**83.8**	47.4	83.7	61.9
T5	**99.5**	62.0	14.4	15.4	89.8	**92.9**	84.8	69.2	41.8	**84.7**	34.7	**84.7**	44.7	**85.2**	59.0
T5+GECA	**99.7**	57.6	10.5	22.8	—	92.5	82.8	66.5	45.8	—	—	—	—	—	—
T5+CSL-Aug.	**99.7**	**100.0**	**99.2**	**99.4**	**99.5**	**93.3**	**89.3**	**74.9**	**67.8**	83.5	**51.6**	83.4	**61.4**	84.0	**70.4**

Table 1: **Main Results.** We compare the performance of our proposed method, T5+CSL-Aug., to prior work across synthetic (SCAN, COGS) and non-synthetic (GeoQuery, SMCalFlow-CS) tasks. Boldfaced results are within 1.0 points of the best result.

GECA experiments are available in Appendix C.4. We also compare with representative prior state-of-the-art methods. For SCAN, NQG-T5 (Shaw et al., 2021) is one of several specialized models that achieves 100% accuracy across multiple splits (Chen et al., 2020; Liu et al., 2020; Nye et al., 2020; Herzig and Berant, 2021). For COGS, we show results from LeAR (Liu et al., 2021), the previously reported state-of-the-art on COGS.[18] We also report new results for NQG-T5 on COGS. For GeoQuery, we report results for NQG-T5[19] and SpanBasedSP (Herzig and Berant, 2021) on the GeoQuery splits we study.[20] For SMCalFlow-CS, we show the strongest previously reported results by Yin et al. (2021), which include the coarse2fine (C2F) model of Dong and Lapata (2018) as a baseline, as well C2F combined with the span-supervised (SS) attention method of Yin et al. (2021). We found it was not computationally feasible to run NQG-T5 on SMCalFlow.

4.3 Main Results

The results are shown in Table 1. For synthetic datasets, the induced grammars have high coverage, making the CSL model highly effective for data augmentation. When we use CSL to generate additional training data for T5 (T5+CSL-Aug.), the performance of T5 improves to nearly solving SCAN and achieving state-of-the-art on COGS.

For non-synthetic tasks, T5+CSL-Aug. leads to new state-of-the-art accuracy on all compositional splits. However, performance is slightly worse on the single-domain splits of SMCalFlow-CS. Based on error analysis in Appendix C.9, we find a significant degree of inherent ambiguity for the remaining errors on the single-domain split, which may contribute to this result.

Using CSL for data augmentation outperforms using GECA on SCAN and GeoQuery. We did not find it computationally feasible to run GECA on COGS or SMCalFlow-CS. On some splits, using GECA to augment the training data for T5 can lead to worse performance, as GECA can over-generate incorrect examples. We provide further analysis comparing CSL and GECA in Appendix C.4.

4.4 Analysis and Discussion

The performance of T5+CSL-Aug. is dependent on the CSL grammar backbone, parametric model, and data sampling details. We analyze the accuracy of T5+CSL-Aug. in relation to CSL's coverage, and summarize the impact of the design choices in CSL that depart from prior work.

Performance Breakdown CSL provides analyses for and can only sample inputs covered by its grammar $x \in \mathcal{X}_{CSL}$, which is often a strict subset of all possible utterances. It is therefore interesting to see how data augmentation impacts T5's performance on covered and non-covered inputs.

In Table 2, we analyze the relative performance of T5, CSL, and combinations of T5 and CSL using ensembling and data augmentation, for non-synthetic compositional splits, partitioning inputs based on whether they are covered by CSL (the same analysis for all splits can be found in Ap-

T5-3B.

[18] We do not show LeAR results for SCAN and GeoQuery as Liu et al. (2021) did not report results for SCAN and reported GeoQuery results using a different template split and a different evaluation metric.

[19] Some of our results for NQG-T5 are different than those reported in Shaw et al. (2021) as we average over 3 new GeoQuery template and TMCD splits, as described in §4.1.

[20] We report exact match accuracy for SpanBasedSP, as opposed to denotation accuracy reported in Herzig and Berant (2021).

		$x \in \mathcal{X}_{CSL}$			$x \notin \mathcal{X}_{CSL}$			All			
Dataset	$\%^{\mathcal{X}_{CSL}}$	T5	CSL	Aug.	T5	CSL	Aug.	T5	CSL	Ens.	Aug.
GeoQuery Templ.	61.0	93.1	96.6	**97.1**	71.6	0.0	**76.9**	84.8	58.9	86.9	**89.3**
GeoQuery TMCD	44.3	88.4	90.3	**93.9**	53.8	0.0	**59.9**	69.2	39.9	70.0	**74.9**
GeoQuery Length	29.0	51.2	**91.6**	83.6	35.4	0.0	**61.3**	40.0	26.6	51.7	**67.8**
SMCalFlow-CS 8-C	6.6	52.3	**79.6**	72.7	33.4	0.0	**50.1**	34.7	5.3	36.5	**51.6**
SMCalFlow-CS 16-C	11.6	59.7	**84.4**	**84.4**	42.7	0.0	**58.4**	44.7	9.8	47.5	**61.4**
SMCalFlow-CS 32-C	13.0	74.4	87.2	**88.4**	56.7	0.0	**67.8**	59.0	11.3	60.6	**70.4**

Table 2: We compare T5, CSL used as a parsing model, and T5+CSL-Aug. (abbreviated as Aug.). We partition evaluation examples by whether CSL generates an output ($x \in \mathcal{X}_{CSL}$) or not ($x \notin \mathcal{X}_{CSL}$) for a given input. The percentage of examples in the former subset ($\%^{\mathcal{X}_{CSL}}$) is limited by the coverage of the induced grammar (see Section 3.1.5). We also compare with an ensemble of T5 and CSL (Ens.) similar to that of Shaw et al. (2021), where we use the output of CSL if $x \in \mathcal{X}_{CSL}$ and otherwise fall back to T5.

pendix C.1). An ensemble model can help T5 only when $x \in \mathcal{X}_{CSL}$, but we can see from Table 2 that data augmentation improves model performance even on inputs not covered by the grammar. For example, for the GeoQuery Length split, performance on non-covered inputs improves from 35.3 to 60.9. This means that T5 is generalizing from the sampled data ($x \in \mathcal{X}_{CSL}$) to $x \notin \mathcal{X}_{CSL}$.

Comparison with NQG We cannot compare using CSL for data augmentation directly with using its closely related predecessor NQG (Shaw et al., 2021) for data augmentation, as NQG is a discriminative parsing model and not a probabilistic generative model that enables sampling new examples. However, we include comparisons of the novel components of CSL relative to the related components of NQG in the following sections, which analyze CSL's grammar induction algorithm and parametric model.

Grammar Induction The grammar induction algorithm of CSL is significantly more scalable than that of NQG, enabling more than 90% decrease in runtime for GeoQuery, and enabling induction to scale to larger datasets such as SMCalFlow-CS. CSL can also induce higher coverage grammars than NQG in some cases, while maintaining high precision. For example, for COGS, the grammar induced by CSL can derive 99.9% of the evaluation set while the grammar induced by NQG can only derive 64.9%. Appendix C.3 contains further analysis comparing the grammar induction algorithms of CSL and NQG. Of course, the grammars induced by CSL can still lack coverage for some datasets, as shown in Table 2. We analyze the limitations of QCFGs in Appendix C.5.

Parameteric Model We find that the simple parameteric model of CSL performs comparably in terms of parsing accuracy to the BERT-based discriminative model of NQG given the same grammar (see Appendix C.3). It is also more scalable because it does not require the computation of a partition function. The variable number of state clusters (§3.1.2) provides a powerful knob for tuning the amount of context sensitivity (see Appendix C.2) to sufficiently fit the training data while also extrapolating to out-of-distribution compositions. We believe further improvements to the parametric model (e.g. using pre-trained representations) have strong potential to improve overall accuracy.

Sampling Results on most splits are significantly improved by using CSL's parametric model compared to sampling uniformly from the induced grammar (Appendix C.6), pointing to a potential source of gains over unweighted augmentation approaches like GECA. However, for SMCalFlow-CS, a higher sampling temperature can improve performance, especially on the 8-shot split, as it leads to $>$ 15 times the number of cross-domain examples being sampled, given their low percentage in the training data. Determining improved methods for biasing the sampling towards examples most relevant to improving performance on the target distribution is an important direction. In Appendix C.7 we explore a setting where we assume access to unlabeled examples from the target distribution, and use these to update the parametric model. We find that this improves sample efficiency with respect to the number of sampled synthetic examples, but can have minimal effect when a sufficiently large number of examples can be sampled. We believe this is a promising research direction.

5 Related Work

Grammar Induction Before the trend towards sequence-to-sequence models, significant prior work in semantic parsing explored inducing SCFG (Wong and Mooney, 2006, 2007; Andreas et al., 2013) and CCG (Zettlemoyer and Collins, 2005, 2007; Kwiatkowksi et al., 2010; Kwiatkowski et al., 2013; Artzi et al., 2014) grammars of the input-output pairs. SCFGs have also been applied to machine translation (Chiang, 2007; Blunsom et al., 2008; Saers et al., 2013). Compression-based objectives similar to ours have also been applied to CFG induction (Grünwald, 1995). Recently, the method of Kim (2021) learns neural parameterized QCFG grammars, which can avoid the pitfalls in coverage of lexicalized grammars such as the ones we learn; however the approach can be computationally demanding for longer input-output pairs.

Data Augmentation Data augmentation has been widely used for semantic parsing and related tasks (Jia and Liang, 2016; Andreas, 2020; Akyürek et al., 2021; Wang et al., 2021b; Zhong et al., 2020; Oren et al., 2021; Tran and Tan, 2020; Guo et al., 2020, 2021). Jia and Liang (2016) perform data recombination using an induced SCFG but their approach requires domain-specific heuristics. GECA (Andreas, 2020) provides a more general solution, which we analyzed in §4.4. The method of Akyürek et al. (2021) is appealing because it can learn data recombinations without committing to a grammar formalism, although gains were limited relative to symbolic methods. The recombination approach of Guo et al. (2020) demonstrates gains for translation tasks but is not as effective as GECA for semantic parsing tasks. Other approaches leverage a forward semantic parser and a backward input generator with some variants (Wang et al., 2021b; Zhong et al., 2020; Tran and Tan, 2020; Guo et al., 2021), but most of these approaches do not explicitly explore the compositional generalization setting. Oren et al. (2021) propose an approach to sample more structurally-diverse data to improve compositional generalization, given a manually specified SCFG.

Compositional Generalization Beyond data augmentation, many approaches have been pursued to improve compositional generalization in semantic parsing, including model architectures (Li et al., 2019; Russin et al., 2019; Gordon et al., 2020; Liu et al., 2020; Nye et al., 2020; Chen et al., 2020; Zheng and Lapata, 2020; Oren et al., 2020; Herzig and Berant, 2021; Ruiz et al., 2021; Wang et al., 2021a), different Transformer variations (Csordás et al., 2021; Ontanón et al., 2021), ensemble models (Shaw et al., 2021), intermediate representations (Herzig et al., 2021), meta-learning (Lake, 2019; Conklin et al., 2021; Zhu et al., 2021), and auxiliary objectives to bias attention in encoder-decoder models (Yin et al., 2021; Jiang and Bansal, 2021). Also, Furrer et al. (2020) compared pre-trained models with specialized architectures.

6 Conclusion

We showed that the Compositional Structure Learner (CSL) generative model improves the state of the art on compositional generalization challenges for two real-world semantic parsing datasets when used to augment the task training data for the generic pre-trained T5 model. Data augmentation using CSL was also largely sufficient to distill CSL's knowledge about compositional structures into T5 for multiple synthetic compositional generalization evaluations. While CSL has limitations (notably, the QCFG formalism is not a good fit for all phenomena in the mapping of natural language to corresponding logical forms), our experiments suggest the strong potential of more powerful probabilistic models over automatically induced latent structures as data generators for black-box pretrained sequence-to-sequence models.

Acknowledgements

We thank Nitish Gupta, Ming-Wei Chang, William Cohen, Pengcheng Yin, Waleed Ammar, the Google Research Language team, and the anonymous reviewers for helpful comments and discussions.

Ethical Considerations

This paper proposed methods to improve compositional generalization in semantic parsing. While we hope that improvements in compositional generalization would lead to systems that generalize better to languages not well represented in small training sets, in this work we have only evaluated our methods on semantic parsing datasets in English.

References

Alfred V Aho and Jeffrey D Ullman. 1972. *The theory of parsing, translation, and compiling*, volume 1. Prentice-Hall Englewood Cliffs, NJ.

Ekin Akyürek, Afra Feyza Akyürek, and Jacob Andreas. 2021. Learning to recombine and resample data for compositional generalization. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net.

Jacob Andreas. 2020. Good-enough compositional data augmentation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7556–7566, Online. Association for Computational Linguistics.

Jacob Andreas, John Bufe, David Burkett, Charles Chen, Josh Clausman, Jean Crawford, Kate Crim, Jordan DeLoach, Leah Dorner, Jason Eisner, Hao Fang, Alan Guo, David Hall, Kristin Hayes, Kellie Hill, Diana Ho, Wendy Iwaszuk, Smriti Jha, Dan Klein, Jayant Krishnamurthy, Theo Lanman, Percy Liang, Christopher H. Lin, Ilya Lintsbakh, Andy McGovern, Aleksandr Nisnevich, Adam Pauls, Dmitrij Petters, Brent Read, Dan Roth, Subhro Roy, Jesse Rusak, Beth Short, Div Slomin, Ben Snyder, Stephon Striplin, Yu Su, Zachary Tellman, Sam Thomson, Andrei Vorobev, Izabela Witoszko, Jason Wolfe, Abby Wray, Yuchen Zhang, and Alexander Zotov. 2020. Task-oriented dialogue as dataflow synthesis. *Transactions of the Association for Computational Linguistics*, 8:556–571.

Jacob Andreas, Andreas Vlachos, and Stephen Clark. 2013. Semantic parsing as machine translation. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 47–52, Sofia, Bulgaria. Association for Computational Linguistics.

Yoav Artzi, Dipanjan Das, and Slav Petrov. 2014. Learning compact lexicons for CCG semantic parsing. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1273–1283, Doha, Qatar. Association for Computational Linguistics.

Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. 2018. Relational inductive biases, deep learning, and graph networks. *ArXiv preprint*, abs/1806.01261.

Phil Blunsom, Trevor Cohn, and Miles Osborne. 2008. A discriminative latent variable model for statistical machine translation. In *Proceedings of ACL-08: HLT*, pages 200–208, Columbus, Ohio. Association for Computational Linguistics.

Peter F. Brown, Stephen A. Della Pietra, Vincent J. Della Pietra, and Robert L. Mercer. 1993. The mathematics of statistical machine translation: Parameter estimation. *Computational Linguistics*, 19(2):263–311.

Xinyun Chen, Chen Liang, Adams Wei Yu, Dawn Song, and Denny Zhou. 2020. Compositional generalization via neural-symbolic stack machines. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*.

David Chiang. 2007. Hierarchical phrase-based translation. *Computational Linguistics*, 33(2):201–228.

John Cocke. 1969. *Programming languages and their compilers: Preliminary notes*. New York University.

Henry Conklin, Bailin Wang, Kenny Smith, and Ivan Titov. 2021. Meta-learning to compositionally generalize. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 3322–3335, Online. Association for Computational Linguistics.

Róbert Csordás, Kazuki Irie, and Juergen Schmidhuber. 2021. The devil is in the detail: Simple tricks improve systematic generalization of transformers. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 619–634, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Li Dong and Mirella Lapata. 2018. Coarse-to-fine decoding for neural semantic parsing. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 731–742, Melbourne, Australia. Association for Computational Linguistics.

Catherine Finegan-Dollak, Jonathan K. Kummerfeld, Li Zhang, Karthik Ramanathan, Sesh Sadasivam, Rui Zhang, and Dragomir Radev. 2018. Improving text-to-SQL evaluation methodology. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 351–360, Melbourne, Australia. Association for Computational Linguistics.

Daniel Furrer, Marc van Zee, Nathan Scales, and Nathanael Schärli. 2020. Compositional generalization in semantic parsing: Pre-training vs. specialized architectures. *ArXiv preprint*, abs/2007.08970.

Jonathan Gordon, David Lopez-Paz, Marco Baroni, and Diane Bouchacourt. 2020. Permutation equivariant models for compositional generalization in language. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net.

Peter Grünwald. 1995. A minimum description length approach to grammar inference. In *International Joint Conference on Artificial Intelligence*, pages 203–216. Springer.

Peter Grunwald. 2004. A tutorial introduction to the minimum description length principle. *arXiv preprint math/0406077*.

Demi Guo, Yoon Kim, and Alexander Rush. 2020. Sequence-level mixed sample data augmentation. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 5547–5552, Online. Association for Computational Linguistics.

Yinuo Guo, Hualei Zhu, Zeqi Lin, Bei Chen, Jian-Guang Lou, and Dongmei Zhang. 2021. Revisiting iterative back-translation from the perspective of compositional generalization. In *AAAI*.

Jonathan Herzig and Jonathan Berant. 2019. Don't paraphrase, detect! rapid and effective data collection for semantic parsing. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3810–3820, Hong Kong, China. Association for Computational Linguistics.

Jonathan Herzig and Jonathan Berant. 2021. Span-based semantic parsing for compositional generalization. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 908–921, Online. Association for Computational Linguistics.

Jonathan Herzig, Peter Shaw, Ming-Wei Chang, Kelvin Guu, Panupong Pasupat, and Yuan Zhang. 2021. Unlocking compositional generalization in pre-trained models using intermediate representations. *ArXiv preprint*, abs/2104.07478.

Theo MV Janssen and Barbara H Partee. 1997. Compositionality. In *Handbook of logic and language*, pages 417–473. Elsevier.

Robin Jia and Percy Liang. 2016. Data recombination for neural semantic parsing. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 12–22, Berlin, Germany. Association for Computational Linguistics.

Yichen Jiang and Mohit Bansal. 2021. Inducing transformer's compositional generalization ability via auxiliary sequence prediction tasks. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 6253–6265, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Mark Johnson. 1998. PCFG models of linguistic tree representations. *Computational Linguistics*, 24(4):613–632.

T. Kasami. 1965. An efficient recognition and syntax analysis algorithm for context-free languages. Technical Report AFCRL-65-758, Air Force Cambridge Research Laboratory, Bedford, MA.

Rohit J Kate, Yuk Wah Wong, and Raymond J Mooney. 2005. Learning to transform natural to formal languages. In *Proceedings of the National Conference on Artificial Intelligence*, volume 20, page 1062. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999.

Daniel Keysers, Nathanael Schärli, Nathan Scales, Hylke Buisman, Daniel Furrer, Sergii Kashubin, Nikola Momchev, Danila Sinopalnikov, Lukasz Stafiniak, Tibor Tihon, Dmitry Tsarkov, Xiao Wang, Marc van Zee, and Olivier Bousquet. 2020. Measuring compositional generalization: A comprehensive method on realistic data. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net.

Najoung Kim and Tal Linzen. 2020. COGS: A compositional generalization challenge based on semantic interpretation. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 9087–9105, Online. Association for Computational Linguistics.

Yoon Kim. 2021. Sequence-to-sequence learning with latent neural grammars. In *Advances in Neural Information Processing Systems*.

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

Dan Klein and Christopher D. Manning. 2003. Accurate unlexicalized parsing. In *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics*, pages 423–430, Sapporo, Japan. Association for Computational Linguistics.

Tom Kwiatkowksi, Luke Zettlemoyer, Sharon Goldwater, and Mark Steedman. 2010. Inducing probabilistic CCG grammars from logical form with higher-order unification. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 1223–1233, Cambridge, MA. Association for Computational Linguistics.

Tom Kwiatkowski, Eunsol Choi, Yoav Artzi, and Luke Zettlemoyer. 2013. Scaling semantic parsers with on-the-fly ontology matching. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1545–1556, Seattle, Washington, USA. Association for Computational Linguistics.

Brenden M. Lake. 2019. Compositional generalization through meta sequence-to-sequence learning. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 9788–9798.

Brenden M. Lake and Marco Baroni. 2018. Generalization without systematicity: On the compositional skills of sequence-to-sequence recurrent networks. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 2879–2888. PMLR.

Brenden M Lake, Tomer D Ullman, Joshua B Tenenbaum, and Samuel J Gershman. 2017. Building machines that learn and think like people. *Behavioral and brain sciences*, 40.

Yuanpeng Li, Liang Zhao, Jianyu Wang, and Joel Hestness. 2019. Compositional generalization for primitive substitutions. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 4293–4302, Hong Kong, China. Association for Computational Linguistics.

Percy Liang. 2013. Lambda dependency-based compositional semantics. *ArXiv preprint*, abs/1309.4408.

Chenyao Liu, Shengnan An, Zeqi Lin, Qian Liu, Bei Chen, Jian-Guang Lou, Lijie Wen, Nanning Zheng, and Dongmei Zhang. 2021. Learning algebraic recombination for compositional generalization. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pages 1129–1144, Online. Association for Computational Linguistics.

Qian Liu, Shengnan An, Jian-Guang Lou, Bei Chen, Zeqi Lin, Yan Gao, Bin Zhou, Nanning Zheng, and Dongmei Zhang. 2020. Compositional generalization by learning analytical expressions. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*.

Richard Montague. 1970. Universal grammar. *Theoria*, 36(3):373–398.

Benjamin Newman, John Hewitt, Percy Liang, and Christopher D. Manning. 2020. The EOS decision and length extrapolation. In *Proceedings of the Third BlackboxNLP Workshop on Analyzing and Interpreting Neural Networks for NLP*, pages 276–291, Online. Association for Computational Linguistics.

Maxwell I. Nye, Armando Solar-Lezama, Josh Tenenbaum, and Brenden M. Lake. 2020. Learning compositional rules via neural program synthesis. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*.

Santiago Ontanón, Joshua Ainslie, Vaclav Cvicek, and Zachary Fisher. 2021. Making transformers solve compositional tasks. *ArXiv preprint*, abs/2108.04378.

Inbar Oren, Jonathan Herzig, and Jonathan Berant. 2021. Finding needles in a haystack: Sampling structurally-diverse training sets from synthetic data for compositional generalization. In *EMNLP*.

Inbar Oren, Jonathan Herzig, Nitish Gupta, Matt Gardner, and Jonathan Berant. 2020. Improving compositional generalization in semantic parsing. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 2482–2495, Online. Association for Computational Linguistics.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21:1–67.

Jorma Rissanen. 1978. Modeling by shortest data description. *Automatica*, 14(5):465–471.

Luana Ruiz, Joshua Ainslie, and Santiago Ontañón. 2021. Iterative decoding for compositional generalization in transformers. *ArXiv preprint*, abs/2110.04169.

Jake Russin, Jason Jo, Randall C O'Reilly, and Yoshua Bengio. 2019. Compositional generalization in a deep seq2seq model by separating syntax and semantics. *ArXiv preprint*, abs/1904.09708.

Markus Saers, Karteek Addanki, and Dekai Wu. 2013. Unsupervised transduction grammar induction via minimum description length. In *Proceedings of the Second Workshop on Hybrid Approaches to Translation*, pages 67–73, Sofia, Bulgaria. Association for Computational Linguistics.

Peter Shaw, Ming-Wei Chang, Panupong Pasupat, and Kristina Toutanova. 2021. Compositional generalization and natural language variation: Can a semantic parsing approach handle both? In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 922–938, Online. Association for Computational Linguistics.

David Smith and Jason Eisner. 2006. Quasi-synchronous grammars: Alignment by soft projection of syntactic dependencies. In *Proceedings on the Workshop on Statistical Machine Translation*, pages 23–30, New York City. Association for Computational Linguistics.

Lappoon R Tang and Raymond J Mooney. 2001. Using multiple clause constructors in inductive logic programming for semantic parsing. In *European Conference on Machine Learning*, pages 466–477. Springer.

Ke Tran and Ming Tan. 2020. Generating synthetic data for task-oriented semantic parsing with hierarchical representations. In *Proceedings of the Fourth Workshop on Structured Prediction for NLP*, pages

17–21, Online. Association for Computational Linguistics.

Bailan Wang, Mirella Lapata, and Ivan Titov. 2021a. Structured reordering for modeling latent alignments in sequence transduction. *Advances in Neural Information Processing Systems*, 34.

Bailin Wang, Wenpeng Yin, Xi Victoria Lin, and Caiming Xiong. 2021b. Learning to synthesize data for semantic parsing. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2760–2766, Online. Association for Computational Linguistics.

Yuk Wah Wong and Raymond Mooney. 2006. Learning for semantic parsing with statistical machine translation. In *Proceedings of the Human Language Technology Conference of the NAACL, Main Conference*, pages 439–446, New York City, USA. Association for Computational Linguistics.

Yuk Wah Wong and Raymond Mooney. 2007. Learning synchronous grammars for semantic parsing with lambda calculus. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 960–967, Prague, Czech Republic. Association for Computational Linguistics.

Pengcheng Yin, Hao Fang, Graham Neubig, Adam Pauls, Emmanouil Antonios Platanios, Yu Su, Sam Thomson, and Jacob Andreas. 2021. Compositional generalization for neural semantic parsing via span-level supervised attention. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2810–2823, Online. Association for Computational Linguistics.

Daniel H Younger. 1967. Recognition and parsing of context-free languages in time n3. *Information and control*, 10(2):189–208.

John M Zelle and Raymond J Mooney. 1996. Learning to parse database queries using inductive logic programming. In *Proceedings of the thirteenth national conference on Artificial intelligence-Volume 2*, pages 1050–1055.

Luke Zettlemoyer and Michael Collins. 2007. Online learning of relaxed CCG grammars for parsing to logical form. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 678–687, Prague, Czech Republic. Association for Computational Linguistics.

Luke S Zettlemoyer and Michael Collins. 2005. Learning to map sentences to logical form: structured classification with probabilistic categorial grammars. In *Proceedings of the Twenty-First Conference on Uncertainty in Artificial Intelligence*, pages 658–666. AUAI Press.

Hao Zheng and Mirella Lapata. 2020. Compositional generalization via semantic tagging. *ArXiv preprint*, abs/2010.11818.

Victor Zhong, Mike Lewis, Sida I. Wang, and Luke Zettlemoyer. 2020. Grounded adaptation for zero-shot executable semantic parsing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6869–6882, Online. Association for Computational Linguistics.

Wang Zhu, Peter Shaw, Tal Linzen, and Fei Sha. 2021. Learning to generalize compositionally by transferring across semantic parsing tasks. *ArXiv preprint*, abs/2111.05013.

Appendix

The appendix is organized into three sections:

- Appendix A contains dataset and preprocessing details.
- Appendix B contains modeling details and hyperparameters.
- Appendix C contains additional experiments and analysis.

A Dataset and Preprocessing Details

Dataset	Split	Train	Dev	Test
SCAN	Jump	14670	—	7706
	Turn Left	21890	—	1208
	Length	11990	—	3920
	MCD1	8365	1046	1045
	MCD2	8365	1046	1045
	MCD3	8365	1046	1045
COGS	Gen	24K	12K	12K
GeoQuery	Standard	600	—	280
	Template1	438	110	332
	Template2	439	110	331
	Template3	440	110	330
	TMCD1	440	110	330
	TMCD2	440	110	330
	TMCD3	440	110	330
	Length	440	110	330
SMCalFlow-CS	8-shot	25412	1324	1325
	16-shot	25420	1324	1325
	32-shot	25436	1324	1325

Table 3: Sizes of all datasets and splits.

In this section we detail preprocessing for each dataset. Dataset sizes are reported in Table 3. We show examples of each dataset in Table 4, with examples of the corresponding induced QCFG rules in Table 5. For each dataset, we report exact match accuracy. We note that all datasets include English language data only; evaluating and extending our method for other languages is an important future direction. We use the same dataset preprocessing for the T5, T5+GECA, NQG-T5, and T5+CSL-Aug. results we report.

SCAN We did not perform any preprocessing for SCAN. Grammar induction does not use any seed rules, and we do not assume a CFG defining valid output constructions, as the outputs consist of action sequences, not executable programs or logical forms.

COGS For COGS, as QCFGs do not support logical variables (Wong and Mooney, 2007), we mapped the original logical forms to a variable-free representation, with an example shown in Table 4. The mapping is deterministic and reversible, and is akin to the use of other variable-free logical forms for semantic parsing such as FunQL (Kate et al., 2005) or Lambda-DCS (Liang, 2013). An alternative but potentially more complex solution to handling logical variables in outputs would be to use an extension of SCFGs, such as λ-SCFG (Wong and Mooney, 2007).

We define an output CFG based on the definition of this variable-free representation. To minimize the linguistic prior, we did not distinguish the types of primitives (e.g., nouns vs verbs); they all belong to the same CFG category. We use a set of seed rules of the form $NT \rightarrow \langle x', x \rangle$ where x is a token found in a training output, and x' is x or an inflected form of x found in a training input (e.g., for $x =$ "sleep", we add $NT \rightarrow \langle \text{sleep}, \text{sleep} \rangle$ and $NT \rightarrow \langle \text{slept}, \text{sleep} \rangle$). These $\langle x', x \rangle$ pairs were identified by running the IBM I alignment model (Brown et al., 1993) on the training data.

GeoQuery We use the same variant of FunQL (Kate et al., 2005) as Shaw et al. (2021), with entities replaced with placeholder values. We generate new length, template, and TMCD splits following the methodology of Shaw et al. (2021), so that we could evaluate our method on dev sets, which the original splits did not include. Specifically, for the length split, we randomly split the test set of the original length split into a dev set of 110 examples and a test set of 330 examples. To reduce variance, we created 3 new template and TMCD splits with different random seeds, with (approximately, in the case of template splits) 440 training examples, and 440 examples that are then randomly split into a 110 dev set and 330 test set. For the TMCD splits, we changed the atom constraint slightly, based on the error analysis in Shaw et al. (2021) which found that a disproportionate amount of the errors on the TMCD test set were in cases where an "atom" was seen in only a single context during training. To create a fairer evaluation of compositional generalization, we strengthen the atom constraint such that every atom in the test set must be seen at least 2 times in the training set. Additionally, as several function symbols in FunQL can be used with and without arguments, and these usages

Dataset	Example
SCAN	x: walk around right and jump thrice y: RTURN WALK RTURN WALK RTURN WALK RTURN WALK JUMP JUMP JUMP
COGS	x: Camila gave a cake in a storage to Emma . y: give (agent = Camila , theme = cake (nmod . in = storage) , recipient = Emma)
GeoQuery	x: what states border states that the m0 runs through y: answer (intersection (state , next_to_2 (intersection (state , traverse_1 (m0)))))
SMCalFlow-CS	x: create work meeting with my boss y: (Yield :output (CreateCommitEventWrapper :event (CreatePreflightEventWrapper :constraint (Constraint[Event] :attendees (AttendeeListHasRecipient :recipient (FindManager :recipient (toRecipient (CurrentUser)))) :subject (?= # (String " work meeting "))))))

Table 4: Example inputs, x, and outputs, y.

Dataset	Induced Rules
SCAN	NT $\rightarrow$ $\langle$NT$_{[1]}$ and NT$_{[2]}$, NT$_{[1]}$ NT$_{[2]}$$\rangle$ NT $\rightarrow$ $\langle$NT$_{[1]}$ thrice, NT$_{[1]}$ NT$_{[1]}$ NT$_{[1]}$$\rangle$ NT $\rightarrow$ $\langle$NT$_{[1]}$ around right, RTURN NT$_{[1]}$ RTURN NT$_{[1]}$ RTURN NT$_{[1]}$ RTURN NT$_{[1]}$$\rangle$
COGS	NT $\rightarrow$ $\langle$NT$_{[1]}$ NT$_{[2]}$ NT$_{[3]}$ NT$_{[4]}$, NT$_{[2]}$ (agent = NT$_{[1]}$, theme = NT$_{[3]}$, recipient = NT$_{[4]}$)$\rangle$ NT $\rightarrow$ $\langle$NT$_{[1]}$ NT$_{[2]}$ NT$_{[3]}$, NT$_{[1]}$ (nmod . NT$_{[2]}$ = NT$_{[3]}$)$\rangle$ NT $\rightarrow$ $\langle$NT$_{[1]}$, a NT$_{[1]}$$\rangle$
GeoQuery	NT $\rightarrow$ $\langle$what NT$_{[1]}$ border NT$_{[2]}$, answer (intersection (NT$_{[1]}$, next_to_2 (NT$_{[2]}$)))$\rangle$ NT $\rightarrow$ $\langle$NT$_{[1]}$ NT$_{[2]}$, intersection (NT$_{[1]}$, NT$_{[2]}$)$\rangle$ NT $\rightarrow$ $\langle$that NT$_{[1]}$ runs through, traverse_1 (NT$_{[1]}$)$\rangle$
SMCalFlow-CS	NT $\rightarrow$ $\langle$NT$_{[1]}$ boss, FindManager :recipient (NT$_{[1]}$)$\rangle$ NT $\rightarrow$ $\langle$NT$_{[1]}$with NT$_{[2]}$, CreateCommitEventWrapper :event (CreatePreflightEventWrapper :constraint (Constraint[Event] :attendees ((AttendeeListHasRecipient :recipient (NT$_{[2]}$)) :subject (? = # (NT$_{[1]}$)))))$\rangle$ NT $\rightarrow$ $\langle$create NT$_{[1]}$, (Yield :output (NT$_{[1]}$))$\rangle$

Table 5: Examples of induced grammar rules for each example in Table 4. Rules without non-terminals are omitted for brevity.

are semantically quite different, we treat function symbols used with different numbers of arguments as different atoms.

We define an output CFG based on the definition of the FunQL operators and the primitive types in the geobase database. We use a set of seed rules of the form $NT \rightarrow \langle x, x \rangle$ where x occurs in both the input and output of a training example.

SMCalFlow-CS To construct SMCalFlow-CS, Yin et al. (2021) filtered out examples that require conversational context. We heuristically filtered out 22 more training examples whose programs contain string literals that are not in the inputs.

We use the original LISP programs provided with the dataset as the output representation. We extract seed rules for string literals and numbers that are copied from inputs to outputs, such as person names and meeting subjects. We add 5 seed rules with a single non-terminal on the input side that enable "hallucinating" various program fragments. We construct an output CFG based on the bracketing of LISP programs and a mapping of argument slots to nonterminals.

B Modeling Details

B.1 QCFG Background and Notation

Synchronous context-free grammars (SCFGs) have been used to model the hierarchical mapping between pairs of strings in areas such as compiler theory (Aho and Ullman, 1972) and multiple natural language tasks, e.g., machine translation (Chiang, 2007) and semantic parsing (Wong and Mooney, 2006; Andreas et al., 2013). SCFGs can be viewed as an extension of context-free grammars (CFGs)

$$\begin{tikzcd}
\langle \mathrm{NT}_{\boxed{1}}, \mathrm{NT}_{\boxed{1}} \rangle \arrow[r, "r_a"] \arrow[dr, "r_c"'] & \langle \alpha_a, \beta_a \rangle \arrow[d, "r_b"] \\
 & \langle \alpha_c, \beta_c \rangle
\end{tikzcd}$$

EXAMPLE

r_a = NT → ⟨NT$_{\boxed{1}}$ and NT$_{\boxed{2}}$, NT$_{\boxed{1}}$ NT$_{\boxed{2}}$⟩
r_b = NT → ⟨jump, JUMP⟩
r_c = NT → ⟨jump and NT$_{\boxed{1}}$, JUMP NT$_{\boxed{1}}$⟩

Figure 4: The arrows in the diagram denote expansion of a nonterminal with a rule. When the above ternary relation holds between r_a, r_b, and r_c, such as in the provided example, we will write $r_a \circ r_b \Rightarrow r_c$. The key sub-routine of our grammar induction algorithm, $\text{UNIFY}(r_1, r_2)$, returns the set of rules $\{r_3 | r_2 \circ r_3 \Rightarrow r_1 \vee r_3 \circ r_2 \Rightarrow r_1\}$.

that *synchronously* generate strings in what we will refer to as an input and output language. We write SCFG rules as $NT \rightarrow \langle \alpha, \beta \rangle$, where NT is a non-terminal symbol, and α and β are strings of non-terminal and terminal symbols.

An SCFG rule can be viewed as two CFG rules, $NT \rightarrow \alpha$ and $NT \rightarrow \beta$, with a pairing between the occurrences of non-terminal symbols in α and β. This pairing is indicated by assigning each non-terminal in α and β an index $\in \mathbb{N}$. Non-terminals sharing the same index are called *linked*. Following convention, we denote the index for a non-terminal using a boxed subscript, e.g. $NT_{\boxed{1}}$.

B.2 Model Parameterization Details

Here we provide the complete definition of the $p_\theta(r|r_p, i)$ and $p_\theta(s|r_p, i)$ terms introduced in § 3.1.2:

$$p_\theta(s|r_p, i) = \frac{e^{\theta_{r_p,i,s}}}{\sum_{s' \in \mathcal{S}} e^{\theta_{r_p,i,s'}}} \tag{8}$$

$$p_\theta(r|s) = \frac{e^{\theta_{s,r}}}{\sum_{r' \in \mathcal{G}} e^{\theta_{s,r'}}} \tag{9}$$

where the θs are scalar parameters. For SMCalFlow-CS, where the number of induced rules is large, we approximate the denominator in Eq. 9 by only considering rules used in derivations in the same batch during training.

B.3 Grammar Induction Algorithm Details

In this section we describe the detailed implementation of the grammar induction algorithm introduced in § 3.1.4.

At each step, we process each rule r_c in $\mathcal{G}$ in parallel. First, using a variant of the CKY algorithm, we check if we can just remove r_c without violating the invariant that all examples in $\mathcal{D}$ can be derived by $\mathcal{G}$. If so, we simply remove the rule r_c as this will always decrease $L_{\mathcal{D}}(\mathcal{G})$. Otherwise, we determine a set of candidate *actions*, A, where an action $a \in A$ consists of a rule to add, r_{add}, and a set of rules to remove, R_{remove}. We determine A using the UNIFY operation described in Figure 4. Specifically, we consider each rule returned by $\text{UNIFY}(r_c, r')$ (where r' is any other rule in $\mathcal{G}$) as a potential rule to add, r_{add}. The corresponding set R_{remove} then consists of r_c and any other rule that we determine can be removed if r_{add} is added, without violating the above invariant.

If a CFG defining valid outputs is provided for the task, we ensure that the output string in r_{add} can be generated by the given CFG, for some replacement of the nonterminal symbols with nonterminals from the output CFG, using a variant of the CKY algorithm.

Given these candidate actions, A, we select:

$$a_{max} = \operatorname*{arg\,max}_{a \in A} \; -L_{\mathcal{D}}(\text{EXEC}(\mathcal{G}, a))$$

where $\text{EXEC}(\mathcal{G}, a)$ is an operation that returns a new set of rules $(\mathcal{G} \cup r_{add}) \setminus R_{remove}$.

We then aggregate over the actions, a_{max}, selected for each rule in $\mathcal{G}$, choosing an action only if it improves the objective. Each action is executed by setting $\mathcal{G} \leftarrow \text{EXEC}(\mathcal{G}, a_{max})$. The algorithm completes if no action was selected or if we reach a configurable number of steps.

We optionally partition the dataset into a configurable number of equally sized partitions based on example length. We then run the algorithm sequentially on each partition, starting with the partition containing the shortest examples. During initialization, we only add rules for examples in the first partition. We then add rules corresponding to examples in the next partition once the algorithm completes on the current partition.

B.4 Hyperparameters

We performed a limited amount of hyperparameter tuning based on performance on development sets. As our goal is to develop models that generalize well across multiple types of distribution shifts, we strove to use the same hyperparameters for each split within a dataset.

Dataset	k_α	k_β	k_t	# Partitions
SCAN	0	100	4	16
COGS	1	5	8	1
GeoQuery	4	16	8	1
SMCalFlow-CS	4	16	8	1

Table 6: Hyperparameters for grammar induction.

Grammar Induction For grammar induction, we selected various configuration options by inspecting the data for each dataset, such as the maximum number of nonterminals in a rule and whether we allow repeated nonterminal indexes. We evaluated several configurations for k_α and k_β in Eq. 7 and the relative cost of terminal vs. nonterminal symbols referenced in Eq. 6, which we will refer to here as k_t, during the development of our algorithm. The selected hyperparameters are listed in Table 6.

Parameteric Model For the CSL parameteric model, we selected a learning rate from $[0.01, 0.05, 0.1]$. We selected a number of context states $|\mathcal{S}|$ from $[32, 64]$, except for SCAN where we analyzed a larger number of context states on the MCD splits, as discussed in Appendix C.2. For SCAN we selected $|\mathcal{S}| = 2$ for all splits, except for MCD1 and MCD3, where we found $|\mathcal{S}| = 4$ to give more consistent performance on the dev sets. This was the only case where we used different hyperparameters for different splits of the same dataset.

Sampling We sampled $100,000$ synthetic examples for all datasets. We provide some analysis of the effect of this in Appendix C.7. For COGS, we biased sampling to increase the number of longer examples (as discussed in § 3.2) by setting the bias $\delta = 6$, and otherwise used $\delta = 0$. We limited the maximum recursion depth to 5 for SCAN, 10 for SMCalFlow, and 20 for GeoQuery and COGS.

T5 Fine-Tuning We started with the same configuration for fine-tuning T5 as Shaw et al. (2021). We similarly selected a learning rate from $[1e^{-3}, 1e^{-4}, 1e^{-5}]$ for each dataset. We use learning rate of $1e^{-3}$ for SCAN and SMCalFlow and $1e^{-4}$ for GeoQuery and COGS.

B.5 Training Details

We train the CSL model on 8 V100 GPUs, which takes less than 1.5 hours for all splits. We fine-tune T5 on 32 Cloud TPU v3 cores[21] for 10,000 steps, which takes less than 6 hours for all splits.

C Additional Analysis

C.1 Performance Breakdown

We extend the performance breakdown analysis of §4.4 to all splits, with results reported in Table 7.

C.2 Varying Context Sensitivity

Varying the number of context states $|\mathcal{S}|$ can vary the degree of context sensitivity in the CSL model. This can be important because we want our model to be able to accurately model $p(y|x)$, which we assume is shared between the source and target distributions, but we also want to sample new inputs x that may have low probability under the source distribution due to the novel compositions they contain.

As a step towards understanding the trade-offs related to context sensitivity, we compute the average $\log p(x, y)$ and $\log p(y|x)$ according to CSL models with different number of context clusters $|\mathcal{S}|$. The results are reported in Table 8. The results also let us compare $\log p(x) = \log p(x, y) - \log p(y|x)$.

A constraint on the number of context states $|\mathcal{S}|$ is in some ways similar to a constraint on the number of nonterminal symbols in a conventional SCFG. Notably, for SCAN, writing a SCFG that unambiguously maps inputs to outputs requires 2 unique nonterminal symbols, and we observe that, similarly, $|\mathcal{S}| \geq 2$ is required to reach 100% accuracy on the dev set. We also observe that while the models with larger $|\mathcal{S}|$ fit the training set better, the log likelihood of the dev sets is highest with $|\mathcal{S}|$ in the range between 2 and 4, indicating that the optimal place on the tradeoff curve is not at the extremes. We also note that there is some variance across the different splits for the optimal number of types, with some values leading to less than optimal modeling of $p(y|x)$.

It is also worth noting that, regardless of the number of context states, the structural conditional independence assumptions in our model can be too strong, harming the accuracy of modeling $p(y|x)$. For example, consider a rule for coordination, $\mathrm{NT} \rightarrow \langle \mathrm{NT}_{[1]} \text{ and } \mathrm{NT}_{[2]}, \mathrm{NT}_{[1]} \wedge \mathrm{NT}_{[2]} \rangle$. In our model, we cannot condition the expansion of $\mathrm{NT}_{[2]}$ on the corresponding expansion of $\mathrm{NT}_{[1]}$ or the parent context in which the coordination rule

[21] https://cloud.google.com/tpu/

		$x \in \mathcal{X}_{CSL}$			$x \notin \mathcal{X}_{CSL}$			All			
Dataset	$\%\mathcal{X}_{CSL}$	T5	CSL	Aug.	T5	CSL	Aug.	T5	CSL	Ens.	Aug.
SCAN Jump	100.0	99.5	100.0	99.7	—	—	—	99.5	100.0	100.0	99.7
SCAN Left	100.0	62.0	100.0	100.0	—	—	—	62.0	100.0	100.0	100.0
SCAN Length	100.0	14.4	100.0	99.2	—	—	—	14.4	100.0	100.0	99.2
SCAN MCD	100.0	15.4	100.0	99.4	—	—	—	15.4	100.0	100.0	99.4
COGS Gen.	99.9	89.8	99.6	99.5	40.9	0.0	100.0	89.8	99.5	99.5	99.5
GeoQuery Std.	76.3	97.2	97.3	98.1	78.9	0.0	77.9	92.9	74.3	93.0	93.3
GeoQuery Templ.	61.0	93.1	96.6	97.1	71.6	0.0	76.9	84.8	58.9	86.9	89.3
GeoQuery TMCD	44.3	88.4	90.3	93.9	53.8	0.0	59.9	69.2	39.9	70.0	74.9
GeoQuery Length	29.0	51.2	91.6	83.6	35.4	0.0	61.3	40.0	26.6	51.7	67.8
SMCalFlow-CS 8-S	30.5	96.0	85.6	95.5	79.8	0.0	78.3	84.7	26.1	81.6	83.5
SMCalFlow-CS 8-C	6.6	52.3	79.6	72.7	33.4	0.0	50.1	34.7	5.3	36.5	51.6
SMCalFlow-CS 16-S	29.5	95.9	85.6	95.9	80.1	0.0	78.2	84.7	25.2	81.7	83.4
SMCalFlow-CS 16-C	11.6	59.7	84.4	84.4	42.7	0.0	58.4	44.7	9.8	47.5	61.4
SMCalFlow-CS 32-S	30.4	96.0	88.1	95.5	80.5	0.0	79.0	85.2	26.7	82.8	84.0
SMCalFlow-CS 32-C	13.0	74.4	87.2	88.4	56.7	0.0	67.8	59.0	11.3	60.6	70.4

Table 7: Performance breakdown for all splits, including non-compositional and synthetic splits, in addition to those already presented in Table 2.

		Context States, $\|\mathcal{S}\|$					
Split	Log prob.	1	2	3	4	8	32
mcd1 (train)	$p(x, y)$	-15.14	-12.55	-10.96	-9.60	-9.21	-9.09
mcd1 (train)	$p(y\|x)$	-0.72	0.00	0.00	0.00	0.00	0.00
mcd1 (dev)	$p(x, y)$	-16.49	-15.04	-17.47	**-13.32**	-18.85	-20.61
mcd1 (dev)	$p(y\|x)$	-0.76	0.00	0.00	0.00	0.00	0.00
mcd2 (train)	$p(x, y)$	-14.38	-11.72	-10.62	-10.26	-9.26	-9.13
mcd2 (train)	$p(y\|x)$	-0.58	0.00	0.00	0.00	0.00	0.00
mcd2 (dev)	$p(x, y)$	-17.60	**-14.10**	-19.84	-17.58	-19.45	-21.82
mcd2 (dev)	$p(y\|x)$	-0.97	0.00	-1.99	-0.36	0.00	-0.04
mcd3 (train)	$p(x, y)$	-15.29	-12.56	-10.86	-11.04	-9.11	-9.09
mcd3 (train)	$p(y\|x)$	-0.75	0.00	0.00	0.00	0.00	0.00
mcd3 (dev)	$p(x, y)$	-16.33	-14.96	**-11.25**	-19.09	-19.83	-20.71
mcd3 (dev)	$p(y\|x)$	-0.75	0.00	0.00	-0.10	0.00	0.00

Table 8: We compare the average log probability CSL assigns to examples from the SCAN MCD train and dev sets, for different numbers of context states, $|\mathcal{S}|$.

is applied, in order to capture notions of type agreement. Such limitations are similar to the limitations of conventional PCFGs to sufficiently model structural dependencies for syntactic parsing (Klein and Manning, 2003).

In general, better understanding and optimizing the trade-offs related to context sensitivity for compositional generalization is an important direction for future work.

C.3 Comparing CSL and NQG

CSL and NQG of Shaw et al. (2021) vary across several dimensions, as the two systems use different grammar induction algorithms and different model parameterizations. Here we compare the two approaches across both dimensions independently.

Grammar Induction As discussed in §3.1.4, the largest set of changes to the CSL algorithm from that of NQG were to improve the scalabilty of the induction algorithm, as both algorithms scale super-linearly in both dataset size and the length of input and output strings. The runtime of grammar induction on the GeoQuery standard split on a standard workstation CPU is around 15 minutes for NQG, and < 1 minute for CSL. More importantly, we did not find it feasible to run NQG for SMCalFlow-CS, while CSL enables grammar induction to be completed within 10 hours with parallelization.

CSL also supports QCFG rules with > 2 nonter-

	GeoQuery			COGS
Parsing Model	Templ.	TMCD	Len.	Gen.
BERT + SpanLabel	58.8	41.0	23.3	99.2
CSL Gen. Model	58.9	39.9	26.6	99.4

Table 9: We compare the accuracy of the span-based BERT-Base model of NQG (BERT + SpanLabel) with that of the CSL generative model (when used as a parsing model) for the same induced grammar (induced using CSL), on compositional splits of GeoQuery and COGS.

minals while NQG does not. We found allowing up to 4 nonterminals can improve the coverage of induced grammars for COGS and SMCalFlow-CS, with some example rules shown in Table 5 in Appendix A. Notably, for COGS, the induction algorithm of NQG induced a grammar that can only derive 64.9% of the test set.

Model Parameterization We compare the performance of CSL's simple generative model with that of the span-based model of NQG which uses a BERT-Base encoder in Table 9. Both models use the *same grammar* induced via the CSL algorithm. Overall, the models perform comparably, despite CSL having far fewer parameters (e.g. for GeoQuery, CSL has only 51,200 parameters[22] while NQG has over 110M parameters as it includes a BERT-Base encoder), not leveraging pre-trained neural networks, and being a generative model to support sampling (in contrast to NQG which is a discriminative model). Incorporating pre-trained neural components into a model such as CSL could be a promising future direction.

For SMCalFlow-CS, given a grammar induced by CSL, we found that it can be computationally infeasible to train a discriminative NQG model, due to the need to compute the partition function which sums over all possible derivations of the input. As a generative model, CSL avoids the need to compute a partition function during training.

C.4 Comparison with GECA

Hyperparameters For SCAN, the reported results in Table 1 use a window size of 1. Using a window size of 2 improves performance on the MCD split (24.9% vs. 22.8%) but hurts performance on the other splits. For GeoQuery, we used the default window size of 4 for the GeoQuery experiments. We attempted to run GECA on COGS and SMCalFlow-CS also using the default hyperparameters and did not find it to be computationally tractable. The algorithm's iteration over templates and fragments can become prohibitive for larger-scale datasets.

Training Examples

⟨jump, JUMP⟩
⟨walk, WALK⟩
⟨jump and walk, JUMP WALK⟩

Synthetic Examples (GECA)

⟨walk and walk, WALK WALK⟩
⟨jump and jump, JUMP JUMP⟩

Synthetic Examples (CSL)

⟨walk and walk, WALK WALK⟩
⟨jump and jump, JUMP JUMP⟩
⟨walk and jump, WALK JUMP⟩
⟨walk and walk and jump, WALK WALK JUMP⟩
⟨walk and jump and walk and walk, WALK JUMP WALK WALK⟩
...

Figure 5: We show the set of derivable synthetic examples for GECA (with window size = 2) and CSL, given an illustrative example of training examples. CSL can derive a significantly larger set of synthetic examples than GECA, and also assigns a probability to each derivable example.

Analysis From Table 1 we see that augmenting the training data using CSL outperforms GECA across both synthetic and non-synthetic evaluations. GECA relies on the simple assumption that fragments are interchangeable if they appear in the same context. It is restricted by a pre-defined window size for fragments, and does not support recursion. Figure 5 compares differences in the sets of derivable synthetic examples for a notional set of training examples. In this case, CSL can derive a much larger set of recombinations by inducing the rule $\mathrm{NT} \rightarrow \langle \mathrm{NT}_{[1]} \text{ and } \mathrm{NT}_{[2]},\ \mathrm{NT}_{[1]}\ \mathrm{NT}_{[2]} \rangle$, which can be applied recursively.

C.5 Limitations of QCFGs

The mapping from inputs to outputs in SCAN, COGS, and GeoQuery are all well supported by QCFGs. However, grammars were used to generate the data for SCAN and COGS, so this is perhaps not surprising. While GeoQuery inputs were written by humans, the distribution of queries in the dataset is influenced by the capabilities of the underlying execution engine based on logic programming; the dataset has a large number of nested noun phrases in inputs that map directly to nested

[22] For each rule and non-terminal token in the induced grammar, CSL has a number of parameters equal to the number of context states.

FunQL clauses in outputs.

SMCalFlow The induced grammars have relatively low coverage on SMCalFlow, as shown by $\%^{\mathcal{X}_{CSL}}$ in Table 7, although they are still sufficient to improve the performance of T5. One reason for the low coverage is that inputs in SMCalFlow often reference specific names, locations, and meeting subjects, such as "setup up a sales meeting with Sam and his manager" where "sales meeting" and "Sam" must be copied to the output program as string literals. Sequence-to-sequence models with copy mechanisms or shared input-output vocabularies can handle such copying, but the QCFGs induced by our method do not support generalization to such novel tokens. Extending the method to support such string copying could significantly improve coverage.

Another reason for the low coverage is that the mismatch between the nesting of prepositional phrases in the input (e.g., "at NT" and "with NT") and the corresponding clauses in the output program tree makes it difficult to induce QCFG rules that enable recombination of different prepositional phrases in different contexts.

The induced QCFGs are also limited in other cases, such as their inability to "distribute" over groupings correctly. Since the training data only contains example such as ⟨Jennifer and her boss, (person = "Jennifer") (FindManager (person= "Jennifer"))⟩, the induced rule NT → ⟨NT and her boss, (person = "NT") (FindManager (person= "NT"))⟩, the induced grammar cannot correctly generate test examples like ⟨Jennifer and Elli and their bosses, (person = "Jennifer") (person = "Elli") (FindManager (person= "Jennifer")) (FindManager (person= "Elli"))⟩.

CFQ We also evaluated the feasibility of our approach to improve T5 performance on CFQ (Keysers et al., 2020), a popular synthetic dataset for evaluating compositional generalization. We found it was challenging to induce QCFGs with reasonable coverage for CFQ. First, the SPARQL queries in CFQ contain variables, which are not well supported by QCFGs (Wong and Mooney, 2007). Additionally, the mapping from queries to SPARQL in CFQ requires notions of commutativity (both "M0 edited and directed M1" and "M0 directed and edited M1" will be mapped to "M0 ns:film.director.film M1 . M0 ns:film.editor.film M1") and distributivity (edited in "edited M1 and M2" will appear twice in "?x0 ns:film.editor.film M1 . ?x0 ns:film.editor.film M2") that are also not well supported by QCFGs. Such limitations can potentially be partially overcome by desigining intermediate representations for CFQ (Furrer et al., 2020; Herzig et al., 2021), but a complete solution likely requires an extension to the class of allowable rules in $\mathcal{G}$ beyond those a QCFG formalism supports, such as better support for variables (Wong and Mooney, 2007) and the ability to apply rewriting rules to generated output strings.

C.6 Sampling Temperature

In this section we study the impact of the parametric model on data augmentation. To do this, we consider varying the sampling temperature, applied to $\theta_{t,r}$ prior to normalization. We compare the accuracy of T5+CSL-Aug. for different temperatures for SMCalFlow-CS, and also with sampling from a uniform distribution rather than using the CSL parameteric model for all splits, which can be viewed as using temperature $= \infty$.

Table 10 shows that using the parameteric model outperforms uniform sampling by a large margin on most splits. However, for SMCalFlow-CS, increasing the sampling temperature can lead to improved performance. To help understand why increasing temperature improves performance on the SMCalFlow-CS cross-domain splits, we computed the number of single-domain examples and cross-domain examples in the 100,000 sampled synthetic examples. Sampling from uniform distribution generates on average 17,764 cross-domain examples comparing with sampling from CSL which generates on average 1,114 cross-domain examples. The significant larger number of synthetic cross-domain examples might explain the improvement on cross-domain performance when increasing sampling temperature, especially on the 8-shot split, given the small number of cross-domain examples in the original training data.

C.7 Semi-Supervised Learning

If we have unlabeled data from our target distribution, consisting of inputs only, we can incorporate this data when training our generative model in a straightforward way. Here we propose a new experiment to evaluate a method for semi-supervised learning that leverages such unlabeled examples. We assume that the unlabeled inputs from the development set are available during training.

	SCAN				COGS	GeoQuery				SMCalFlow-CS					
Temp.	Jump	Left	Len.	MCD	Gen.	Std.	Templ.	TMCD	Len.	8-S	8-C	16-S	16-C	32-S	32-C
∞	97.2	98.0	42.0	43.8	91.4	92.9	88.0	72.9	51.5	83.4	63.1	84.4	62.6	83.8	71.5
10	—	—	—	—	—	—	—	—	—	82.8	62.4	83.1	63.8	82.5	71.0
1	99.7	100.0	99.2	99.4	99.5	93.3	89.3	74.9	67.8	83.5	51.6	83.4	61.4	84.0	70.4

Table 10: We compare sampling from CSL and uniform distribution (temperature = ∞) for the same induced grammar.

		GeoQuery				SCAN
Semi.	Example #	Std.	Templ.	TMCD	Len.	MCD
	1K	92.9	89.3	72.5	66.4	63.8
✓	1K	93.2	89.3	74.4	67.9	88.9
	100K	93.3	89.3	74.9	67.8	99.4
✓	100K	93.2	89.4	75.6	67.3	99.4

Table 11: We compare the accuracy of T5 + CSL using data augmentation with and without unlabeled data.

Experiment Setting In this setting, conceptually, when optimizing θ, we want to maximize both the likelihood of $p(x, y)$ for $\mathcal{D}$ and the marginal likelihood $p(x)$ for the unlabeled data. As the latter requires marginalizing over derivations for x and all possible outputs, and this can be a large set to sum over, we approximate this by first labeling the unlabeled data following the inference procedure described in §3.1.5 using the generative model trained only on $\mathcal{D}$, which can be interpreted as a hard-EM approach. During this process, we discard any unlabeled that cannot be derived given $\mathcal{G}$. We then re-train the generative model on both sets of data following the standard procedure, duplicating the "unlabeled" data a configurable number of times to achieve the desired ratio to the original labeled data.

We evaluate our CSL + T5-Aug. in this setting using *unlabeled* data from the development set. We use the SCAN MCD splits as they have dev sets available. We also evaluate performance on the GeoQuery splits. We compare the performance with and without unlabeled data using 1,000 and 100,000 synthetic examples.

Results Results are reported in Table 11. Incorporating unlabeled examples leads to improvements when sampling only 1,000 examples, but leads to minimal improvements when sampling 100,000 examples. We did not find positive results based on initial experiments for SMCalFlow-CS, likely due to the low coverage of the induced grammars on the target examples (see Table 7), as the method we evaluated cannot leverage unlabeled examples that are not covered by the induced grammar.

Split	Mean	Stdev.
Std.	93.3	0.2
Templ.1	92.5	0.3
Templ.2	88.1	0.8
Templ.3	87.2	0.6
TMCD1	77.2	1.1
TMCD2	71.3	0.6
TMCD3	76.3	0.2
Len.	67.8	0.3

Table 12: The mean and standard deviation of 3 runs for T5+CSL-Aug. on GeoQuery dataset.

C.8 GeoQuery Variance

The variance of T5+CSL-Aug. for GeoQuery is reported in Table 12.

C.9 SMCalFlow-CS Error Analysis

We sampled 20 prediction errors for T5+CSL-Aug. from single-domain and cross-domain development sets respectively. We found a large number of errors are due to ambiguous and inconsistent annotations. Table 13 shows some examples of such errors. First, the subject string is determined inconsistently for training and testing examples. Second, the same source can be mapped to different targets which express the same meaning. Third, some examples require additional context to generate the correct output. Among the errors we sample, around 60% of single-domain errors and around 35% of cross-domain errors fall into these three types. In addition, for the cross-domain examples, T5+CSL-Aug. sometimes struggles with nesting programs in a correct way when examples require querying an org chart for more than one people as discussed in Appendix C.5.

Train Source: Can you create an Meeting for Saturday 1 : 00 pm **Train Target:** (Yield :output (CreateCommitEventWrapper :event (CreatePreflightEventWrapper :constraint (Constraint[Event] :start (?= (DateAtTimeWithDefaults :date (NextDOW :dow # (DayOfWeek " SATURDAY ")) :time (NumberPM :number # (Number 1)))) :subject (?= # (String " Meeting ")))))) **Dev Source:** Schedule a meeting on Thursday at 8 : 30 AM . **Dev Target:** (Yield :output (CreateCommitEventWrapper :event (CreatePreflightEventWrapper :constraint (Constraint[Event] :start (?= (DateAtTimeWithDefaults :date (NextDOW :dow # (DayOfWeek " THURSDAY ")) :time (HourMinuteAm :hours # (Number 8) :minutes # (Number 30.0)))))))) **Dev Prediction:** (Yield :output (CreateCommitEventWrapper :event (CreatePreflightEventWrapper :constraint (Constraint[Event] :start (?= (DateAtTimeWithDefaults :date (NextDOW :dow # (DayOfWeek " THURSDAY ")) :time (HourMinuteAm :hours # (Number 8) :minutes # (Number 30.0))))) :subject (?= # (String " meeting "))))))
Train Source: Schedule 3 pm tentative shareholders huddle **Train Target:** (Yield :output (CreateCommitEventWrapper :event (CreatePreflightEventWrapper :constraint (Constraint[Event] :start (?= (NextTime :time (NumberPM :number # (Number 3)))) :subject (?= # (String " tentative shareholders huddle ")))))) **Dev Source:** Schedule 3 pm tentative shareholders huddle **Dev Target:** (Yield :output (CreateCommitEventWrapper :event (CreatePreflightEventWrapper :constraint (Constraint[Event] :showAs (?= # (ShowAsStatus " Tentative ")) :start (?= (NextTime :time (NumberPM :number # (Number 3)))) :subject (?= # (String " shareholders huddle ")))))) **Dev Prediction:** (Yield :output (CreateCommitEventWrapper :event (CreatePreflightEventWrapper :constraint (Constraint[Event] :start (?= (NextTime :time (NumberPM :number # (Number 3)))) :subject (?= # (String " tentative shareholders huddle "))))))
Dev Source: create football game on tuesday at 8 **Dev Target:** (Yield :output (CreateCommitEventWrapper :event (CreatePreflightEventWrapper :constraint (Constraint[Event] :start (?= (DateAtTimeWithDefaults :date (NextDOW :dow # (DayOfWeek " TUESDAY ")) :time (NumberPM :number # (Number 8)))) :subject (?= # (String " football game ")))))) **Dev Prediction:** (Yield :output (CreateCommitEventWrapper :event (CreatePreflightEventWrapper :constraint (Constraint[Event] :start (?= (DateAtTimeWithDefaults :date (NextDOW :dow # (DayOfWeek " TUESDAY ")) :time (NumberAM :number # (Number 8)))) :subject (?= # (String " football game "))))))

Table 13: Example prediction errors for T5+CSL-Aug. and their closest training example if any for the SMCalFlow-CS dataset.